Ejercicios de listas

Ejercicio 1 En ciertas aplicaciones interesa separar las corridas ascendentes en una lista de números $L = (\ a_1,\ a_2,\ ...,\ a_n\ )$, donde cada corrida ascendente es una sublista de números consecutivos $a_i, a_{i+1}, ...,$ a_{i+k}, la cual termina cuando $a_{i+k} > a_{i+k+1}$, y es ascendente en el sentido de que $a_i \leq a_{i+1} \leq ... \leq a_{i+k}$. Por ejemplo, si la lista es L = (0,5,6,9,4,3,9,6,5,5,2,3,7), entonces hay 6 corridas ascendentes, a saber: (0,5,6,9), (4), (3,9), (6), (5,5) y (2,3,7). *Consigna:* usando las operaciones de la clase lista, escribir una función `int ascendente (list <int> &L, list < list<int> > &LL)` en la cual, dada una lista de enteros `L`, almacena cada corrida ascendente como una sublista en la lista de listas `LL`, devolviendo además el número z de corridas ascendentes halladas. *Restricciones:* a) El tiempo de ejecución del algoritmo debe ser $O(n)$, b) La lista de listas `LL` inicialmente está vacía, c) No usar otras estructuras auxiliares. [Tomado en Examen Final 29-JUL-2004].

Solución

```
1  // . los iteradores "p,q" recorren los numeros de la lista "L"
2  // . el iterador "r" va pasando de sublista en sublista, donde
3  //   cada sublista es una corrida ascendente de la lista "L";
4  // . la lista vacia A es solo para inicializar cada nueva sublista.
5  int  ascendente1_a (list <int> &L, list < list<int> > &LL) {
6    list < list<int> > :: iterator r;
7    list <int> :: iterator p, q;
8    list <int> A;
9    p = L.begin ();
10   while (p != L.end ()) {
11     // inserta una nueva sublista vacia "A" en la lista de listas "L"
12     r = LL.insert (LL.end(),A);
13     // inserta el valor "*p" de la lista "L" en la nueva sublista
14     r->insert (r->end(), *p) ;
15     // mientras no sea fin de la lista "L" y se verifica la condicion
16     // de corrida ascendente, va copiando a la sublista actual
17     q = p ; q++;
18     while (q != L.end () && *p <= *q) {
19       r->insert (r->end(), *q) ;
20       p++;
21       q++;
22     } // end while
23     p = q ;   // se posiciona al principio de la sgte corrida
24   } // end while
25   printa (LL) ;
26   return LL.size() ;
27 }
28
29 //--------------------------------------------------------------------
30 // Otra solucion
31 int  ascendente1_b (list<int> &L, list<list<int> > &LL) {
32   list<int>::iterator p, q;
33   list<list<int> >::iterator ql ;
34   int x ;
35   LL.clear();
36   p = L.begin ();
37   while (p != L.end ()) {
38     ql = LL.insert(LL.end(),list<int>());
39     while (1) {
40       x = *p++;
41       ql->insert(ql->end(),x);
42       if (p==L.end() || *p<x) break;
43     } // end while
44   } // end while
45   return LL.size();
46 }
```

Resuelto en el archivo `ascendente.cpp`

Ejercicio 2 Determinar si una cadena `z` es de la forma `z = x y`, donde `y` es la cadena inversa (o espejo) de la cadena `x`, ignorando los espacios en blanco. Emplear una cola y una pila auxiliares.

Solución

```
1  bool cadena_pq (list <char> & L1, list <char> & L2) {
2    queue <char> Q;
3    stack <char> S;
4    list<char>::iterator p;
5    list<char>::iterator q;
6    char x, y ;
7    bool b ;
8
9    // verifica que las longitudes de las listas "L1" e "L2" sean iguales
10   // Esto es dudoso, en algunas implementaciones
11   // el size() es O(n) (la del gcc particularmente)
12   b = ( L1.size() == L2.size () ) ;
13   if (!b) {
14     cout << endl ;
15     cout << "error: listas de longitudes distintas" << endl;
16     return false ;
17   }
18   cout << endl ;
19
20   // pone cada caracter de la cadena "x" en la pila S
21   p = L1.begin();
22   while ( p != L1.end () ) {
23     x = *p ;
24     S.push (x) ;
25     p++;
26   } // end while
27
28   // pone cada caracter de la cadena "y" en la cola Q
29   q = L2.begin();
30   while ( q != L2.end () ) {
31     y = *q;
32     Q.push (y) ;
33     q++;
34   } // end while
35
36   // itera comparando el tope de la pila y el frente de la cola
37   while (!S.empty () && !Q.empty () ) {
38     x = S.top ()   ; S.pop (); // no olvidar "desapilar" !!
39     y = Q.front () ; Q.pop (); // no olvidar "descolar"  !!
40     cout << "tope (pila) = " << x << " ; frente (cola) = " << y << endl;
41     if (x != y) { return false ; }
42   } //
43   return true ;
44 }
```

Resuelto en el archivo `cadena_pq.cpp`

Ejercicio 3 Escribir una función `void chunk_revert(list<int> &L,int n);` que dada una lista `L` y un entero `n`, invierte los elementos de la lista tomados de a `n`. Si la longitud de la lista no es múltiplo de `n` entonces se invierte el resto también. Por ejemplo, si `L=1,3,2,5,4,6,2,7` entonces después de hacer `chunk_revert(L,3)` debe quedar `L=2,3,1,6,4,5,7,2`. Restricciones: Usar a lo sumo una estructura auxiliar. (En tal caso debe ser lista, pila o cola). [Tomado en el 1er parcial 21/4/2005].

Solución

```
1  // Version con pila
2  void chunk_revert (list<int> &L, int n) {
3    stack<int> S;
4    list<int>::iterator q;
5    q = L.begin();
6    while (q!=L.end()) {
7      for (int j=0; j<n; j++) {
8        if (q == L.end()) break;
9        S.push(*q);
10       q = L.erase(q);
11     } // end for
12     while (!S.empty()) {
13       q = L.insert(q,S.top());
14       q++;
15       S.pop();
16     } // end while
17   } // end while
18 }
19
20 // ---------------------------------------------------------------------
21 // Version con cola
22 void chunk_revert4 (list<int> &L, int n) {
23   queue<int> Q;
24   list<int>::iterator q;
25   int m ;
26   q = L.begin();
27   while (q!=L.end()) {
28     for (int j=0; j<n; j++) {
29       if (q == L.end()) break;
30       Q.push(*q);
31       q = L.erase(q);
32     } // end for
33     m = Q.size();
34     while (!Q.empty()) {
35       q = L.insert(q,Q.front());
36       Q.pop();
37     } // end while
38     for (int j=0; j<m; j++) q++;
39   } // end while
40 }
41
42 // ---------------------------------------------------------------------
43 // Version 'in place'
44 void chunk_revert2 (list<int> &L, int n) {
45   list<int>::iterator q, p;
46   q = L.begin();
47   int m, x;
48   while (q!=L.end()) {
49     p = q;
50     for (m=0; m<n; m++)
51       if (p++ == L.end()) break;
52     for (int j=0; j<m; j++) {
53       x = *q;
54       q = L.erase(q);
55       p = q;
56       for (int k=0; k<m-j-1; k++) p++;
57       p = L.insert(p,x);
58     } // end for
59     q = p;
60     for (int j=0; j<m; j++) q++;
61   } // end while
62 }
63
64 // ---------------------------------------------------------------------
```

```
65 // Version 'in place' 2 (sin suprimir ni insertar)
66 void chunk_revert3(list<int> &L,int n) {
67   list<int>::iterator q, p, r;
68   int m, x;
69   q = L.begin();
70   while (q!=L.end()) {
71     p = q;
72     for (m=0; m<n; m++)
73       if (p++==L.end()) break;
74     r = q;
75     for (int j=0; j<m/2; j++) {
76       x = *q;
77       p = q;
78       for (int k=0; k<m-2*j-1; k++) p++;
79       *q = *p;
80       *p = x;
81     } // end for
82     q = r;
83     for (int j=0; j<m; j++) q++;
84   } // end while
85 }
```

Resuelto en el archivo `chunk-revert.cpp`

Ejercicio 4 Escriba procedimientos para concatenanar: a) dos listas `L1` y `L2` usando `insert`; b) un vector `VL` de `n` listas usando `insert`; c) una lista `LL` de `n` sublistas usando `insert` "básico"; d) una lista `LL` de `n` sublistas usando una opción de `insert`; e) una lista `LL` de `n` sublistas usando `splice`.

Solución

```
1  //-------------------------------------------------------------------
2  // concatenana dos listas "L1" y "L2" usando "insert"
3  template <typename t>
4  void concat_insert_2l (list<t> &L1,list<t> &L2,list<t> &L){
5    L.clear();
6    L.insert(L.end(),L1.begin(),L1.end());
7    L.insert(L.end(),L2.begin(),L2.end());
8  }
9
10 //-------------------------------------------------------------------
11 // concatenana el vector"VL" de "n" listas usando "insert"
12 template <typename t>
13 void concat_insert_vl (vector< list<t> > &VL,list<t> &L) {
14   typename list<t>::iterator q,z;
15   int n;
16   L.clear();
17   n=VL.size();
18   for (int i=0;i<n;i++) {
19     q=VL[i].begin();
20     z=VL[i].end();
21     while (q!= z) L.insert(L.end(),*q++);
22   } // end while
23 }
24
25 //-------------------------------------------------------------------
26 // concatenana una lista LL de "n" sublistas usando insert "basico"
27 template <typename t>
28 void concat_inserb_ll(list< list<t> > &LL,list<t> &L) {
29   typename list< list<t> >::iterator p,y;
30   typename list<t>::iterator q,z;
31   L.clear();                          // re-inicializa nueva lista
32   p=LL.begin();                       // iterador de la lista
33   y=LL.end();                         // fin de la lista
34   while (p!=y) {                      // recorre lista
35     q=(*p).begin();                   // *p recorre sublista
36     z=(*p).end();                     // *z es fin de sublista
37     while (q!=z) L.insert(L.end(),*q++);// inserta la sublista actual
38     p++;                              // avanza a la sgte sublista
39   } // end while
40 }
41
42 //-------------------------------------------------------------------
43 // concatenana lista LL de "n" sublistas usando una opcion de insert
44 template <typename t>
45 void concat_interv_ll(list< list<t> > &LL,list<t> &L) {
46   typename list< list<t> >::iterator p;
47   L.clear();
48   for (p=LL.begin();p!=LL.end();p++) {
49     L.insert(L.end(),(*p).begin(),(*p).end());
50   } // end i
51 }
52
53 //-------------------------------------------------------------------
54 // concatenana la lista "LL" de "n" sublistas usando "splice"
55 template <typename t>
56 void concat_splice_ll(list< list<t> > &LL,list<t> &L) {
57   typename list< list<t> >::iterator p;
58   L.clear();
59   for (p=LL.begin();p!=LL.end();p++) L.splice(L.end(),*p);
60 }
```

Resuelto en el archivo `concatena.cpp`

Ejercicio 5 Escribir una función `void creciente(queue<int> &Q)` que elimina elementos de `Q` de tal manera de que los elementos que quedan estén ordenados en forma creciente. [Tomado en el 1er parcial 27-APR-2004]

Solución

```
1  void creciente (queue <int> & Q) {
2    int w, max ;
3    // Asegura que al menos haya un elemento
4    if (Q.empty()) return;
5    // Cola auxiliar
6    queue<int> Q2;
7    // 'max' mantiene el maximo de los elementos
8    // hasta ahora visitados
9    max = Q.front();
10   while (!Q.empty()) {
11     // Saca elemento de 'Q' y lo mantiene en 'w'
12     w = Q.front();
13     Q.pop();
14     if (w >= max) {
15       // Si es >= lo pone en 'Q2' y
16       // actualiza el maximo actual.
17       max = w;
18       Q2.push(w);
19     } // end if
20   } // end while
21   // Pasa todo 'Q2' de vuelta a 'Q'.
22   while (!Q2.empty()) {
23     Q.push (Q2.front());
24     Q2.pop ();
25   }
26 } // end void
```

Resuelto en el archivo `creciente.cpp`

Ejercicio 6 Implemente una función `encuentra(list<int> &L1, list<int> &L2, list<int> &indx)` que verifica si los elementos de 'L2' estan en 'L1' (en el mismo orden, pero no necesariamente en forma consecutiva). Si es asi, retorna true y en 'indx' retorna los indices de los elementos de 'L1' que corresponden a los elementos de 'L2'.

Solución

```
1  bool encuentra(list<int> &L1, list<int> &L2, list<int> &indx) {
2    indx.clear();
3    list<int>::iterator
4    p1=L1.begin(),
5    p2=L2.begin();
6    int j=0;
7    while (p1!=L1.end() && p2!=L2.end()) {
8      if (*p2 == *p1) { p2++; indx.push_back(j); }
9      p1++; j++;
10   } // end while
11   if (p2==L2.end()) return true;
12   else {indx.clear(); return false;}
13 }
```

Resuelto en el archivo `encuentra.cpp`

Ejercicio 7 Escriba procedimientos para intercalar (*merge*): (i) dos listas ordenadas `L1` y `L2` en una nueva lista `L`; (ii) un vector `VL` de `n` listas ordenadas como nueva lista `L`. Notar que *intercalar* (*merge*) implica en ambos casos que la nueva lista `L` debe resultar también *ordenada*.

Solución

```
1  // intercala (merge) dos listas ordenadas L1 y L2 como nueva lista L
2  template <typename t>
3  void intercala_2L (list<t> &L1,list<t> & L2 ,list<t> &L){
4          typename list<t>::iterator p,q;
5          t x1,x2;
6          L.clear();          // reinicializa nueva lista L
7          p=L1.begin();       // iterador para recorrer lista L1
8          q=L2.begin();       // iterador para recorrer lista L2
9          while (p!=L1.end() && q!=L2.end()) {
10                 x1=*p;
11                 x2=*q;
12                 if (x1<=x2) {
13                         L.insert(L.end(),x1);
14                         p++;
15                 }
16                 else {
17                         L.insert(L.end(),x2);
18                         q++;
19                 } // end if
20         } // end while
21         while (p!=L1.end()) {L.insert(L.end(),*p++);} // eventual resto L1
22         while (q!=L2.end()) {L.insert(L.end(),*q++);} // eventual resto L2
23 }
24
25 // -------------------------- -------------------------------------
26 // intercala el vector VL de "n" listas ordenadas como nueva lista L
27 template <typename t>
28 void intercala_vn (vector< list<t> > &VL, list<t> &L){
29         typename list<t>::iterator p,q,z;
30         int n;
31         t x1,x2;
32         L.clear();          // reincializa nueva lista L
33         // copia la primera lista L_0 del vector VL en la lista L
34         n=VL.size();
35         q=VL[0].begin();    // iterador para la lista L_0
36         z=VL[0].end();      // posicion del fin de la lista L_0
37         while (q!=z) L.insert (L.end(),*q++); // notar: inserta al final
38         // ahora intercala las restantes del vector en la lista L
39         for (int i=1;i<n;i++) {
40                 p=L.begin();        // iterador para recorrer la nueva lista L
41                 q=VL[i].begin();    // iterador para recorrer la lista L_i
42                 z=VL[i].end();      // fin de la lista L_i
43                 while (p!=L.end() && q!=z) {
44                         x1=*p;
45                         x2=*q;
46                         if (x1<=x2) {p++;}           // no inserta y avanza
47                         else {p=L.insert(p,x2);q++;} // inserta y refresca posicion
48                 } // end while
49                 // pasa el remanente de la lista L_i a la nueva lista L
50                 while (q!=z) L.insert(L.end(),*q++); // notar: inserta al final
51         } // end i
52 }
```

Resuelto en el archivo `intercala.cpp`

Ejercicio 8 7. Problema de Josephus. Un grupo de soldados se haya rodeado por una fuerza enemiga. No hay esperanzas de victoria si no llegan refuerzos y existe solamente un caballo disponible para el escape. Los soldados se ponen de acuerdo en un pacto para determinar cuál de ellos debe escapar y solicitar ayuda. Forman un círculo y se escoge un número n al azar. Igualmente se escoge el nombre de un soldado. Comenzando por el soldado cuyo nombre se ha seleccionado, comienzan a contar en la dirección del reloj alrededor del círculo. Cuando la cuenta alcanza el valor n, este soldado es retirado del círculo y la cuenta comienza de nuevo, con el soldado siguiente. El proceso continúa de tal manera que cada vez que se llega al valor de n se retira un soldado. El último soldado que queda es el que debe tomar el caballo y escapar. Entonces, dados un número n y una lista de nombres, que es el ordenamiento en el sentido de las agujas del reloj de los soldados en el círculo (comenzando por aquél a partir del cual se inicia la cuenta), escribir un procedimiento que obtenga los nombres de los soldados en el orden que han de ser eliminados y el nombre del soldado que escapa.

Solución

```
// -----------------------------------------------------------------
// Debe retornar una lista con las numeros relativos de soldados de
// que van saliendo segun el algoritmo de Josephus, donde "n" es la
// cantidad de soldados y "s" es el salto en el juego
// -----------------------------------------------------------------
void josephus (int n,int s,list<int> &L) {
  list <int> H;
  list <int> :: iterator p;
  // Inicialmente carga en lista auxiliar H con los enteros [0,n]
  for (int j=0;j<n;j++) H.insert(H.end(),j);
     p=H.begin();
    // Va simulando el algoritmos extrayendo soldados de H y
    // pasandolos a L. Como hay que extraer exactamente N soldados
    // directamente hacemos un lazo de 0 a N-1
    for (int k = 0; k < n ; k++) {
    // Avanzamos S posiciones en sentido circular por lo que nunca
    // debe quedar en H.end (). Para evitarlo, cuando llega a
    // ser H.end () pasamos a H.begin ().
    for (int j = 0 ; j < s-1; j++)
      if (++p == H.end()) p = H.begin (); // Notar pre-incremento
      // Pasamos el soldado en P a la lista L
      L.insert (L.end(),*p);
      // Borra en sentido circular, es decir, si P es el
      // ultimo elemento, entonces al borrar queda en H.end(),
      // en ese caso lo pasamos a H.begin ()
      p = H.erase(p);
      if (p == H.end () ) p = H.begin ();
    } // end j
} // end void
```

Resuelto en el archivo `josephus.cpp`

Ejercicio 9 Escribir una función `void junta (list <int> &L, int n)` que, dada una lista `L`, agrupa de a `n` elementos dejando su suma IN PLACE. Por ejemplo, si la lista `L` contiene L=(1,3,2,4,5,2,2,3,5,7,4,3,2,2), entonces depués de `junta (L,3)` debe quedar L=(6,11,10,14,4). Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución `O (m)`, donde `m` es el número de elementos en la lista original. [Tomado en el examen final del 1/8/2002]

Solución

```
1  void junta(list<int> &L,int n) {
2    list<int>::iterator p;
3    int suma;
4    p=L.begin();
5    while (p!=L.end()) {
6      suma=0;
7      // Suma de N elementos y los elimina
8      for (int j=0;j<n;j++) {
9        suma=suma+(*p);
10       p=L.erase(p);
11       // Atencion podrian quedar menos de N elementos
12       // en la lista. En tal caso llegamos a L.end () y
13       // debemos salir del lazo
14       if (p==L.end()) break;
15     } // end i
16     // Inserta SUMA e incrementa P de manera de quedar apuntando
17     // al comienzo de la siguiente subsecuencia.
18     p=L.insert(p,suma);p++;
19   } // end while
20 }
```

Resuelto en el archivo `junta.cpp`

Ejercicio 10 Escribir una función `void ordenag (list <int> &l, int m)` que, dada una lista `l`, va ordenando sus elementos de a grupos de `m` elementos. Por ejemplo si m=5, entonces `ordenag` ordena los primeros 5 elementos entre si, despues los siguientes 5 elementos, y asi siguiendo. Si la longitud `n` de la lista no es un múltiplo de `m`, entonces los últimos `n mod m` elementos también deben ser ordenados entre si. Por ejemplo, si l = (10 1 15 7 2 19 15 16 11 15 9 13 3 7 6 12 1), entonces después de `ordenag (5)` debemos tener l = (1 2 7 10 15 11 15 15 16 19 3 6 7 9 13 1 12). [Tomado en el examen final del 5-Dic-2002].

Solución

```
1  void  ordenag (list<int> &L,int m) {
2    list <int>::iterator p,q,z;
3    int  k, x;
4    p = L.begin();
5    while (p != L.end()) {
6      for (int k = m ; k > 0 ; k--) {
7        // busca el menor elemento *z= min(*q) en el rango [p,p+k)
8        q = p;
9        z = q;
10       q++;
11       for (int j = 1 ; j < k ; j++) {
12         if (q == L.end()) break;
13         if (*q < *z) z=q;
14         q++;
15       } // end j
16       // lo inserta en L (p) y lo borra de R
17       x =*z;
18       if (z!=p) {
19         L.erase(z);
20         p=L.insert(p,x);
21       } // end if
22       p++;
23       if (p == L.end()) break;
24     } // end k
25   } // end while
26   p=L.begin();
27   k=0;
28   while (p!=L.end()) {
29     cout << *p++ << " ";
30     if (! (++k % m) ) cout << endl;
31   } // enf while
32   cout << endl;
33 } // end void
```

Resuelto en el archivo `ordenag.cpp`

Ejercicio 11 Usando las operaciones del TAD lista, escribir una función `void particiona (list<int> &L, int a)` la cual, dada una lista de enteros `L`, reemplace aquellos que son mayores que `a` por una sucesión de elementos menores o iguales que `a` pero manteniendo la suma total constante. [Ejercicio tomado en el Exámen Final del 05/07/01]

Solución

```
1  void particiona (list<int> &L, int a) {
2    list <int>::iterator p;
3    int x;
4    p = L.begin();
5    while (p != L.end()) {
6      // Despues de cada ejecucion del cuerpo de este lazo
7      // si      *p <= a entonces p queda apuntando al siguiente
8      // pero si *p >  a entonces *p es descompuesto en una serie
9      // valores  y p queda apuntando al elemento siguiente
10     // de la secuencia
11     x=*p;
12     if (x>a) {
13       p=L.erase(p); // Eliminamos el elemento
14       // Este lazo inserta tantos valores de a como entren en *p
15       while (x>a) {
16         p=L.insert(p, a);
17         p++;
18         x=x-a;
19       } // end while
20       // Inserta el resto (si es x > 0)
21       if (x>0) {p=L.insert(p,x); p++;}}
22     else
23       p++;
24   } // end while
25 }
```

Resuelto en el archivo `particiona.cpp`

Ejercicio 12 Escriba una función `void print_back (list<int> & L, list <int>::iterator p)` que, en forma *recursiva*, imprima una lista en sentido inverso, es decir, desde el final al principio de la lista. Se le da como dato el procedimiento a la primera posición de la lista. [Ejercicio 3 del final del 14/02/2002]

Solución

```
1  // imprime en orden inverso (por recursion, de atras para adelante)
2  void print_back (list<int> &L,list<int>::iterator p) {
3    list<int>::iterator q;
4    if (p==L.end()) return;
5    q=p ; q++;
6    print_back (L,q);
7    cout << *p << " ";
8  }
9  void print_back (list<int> &L) {
10   print_back (L,L.begin());
11   cout << endl;
12 }
```

Resuelto en el archivo `print_back.cpp`

Ejercicio 13 Usando las operaciones del TAD lista, escribir una función `void random_shuffle (list <int> &L)` que, dada una lista de enteros `L`, reordena sus elementos en forma aleatoria. Se sugiere el siguiente algoritmo: usando una lista auxiliar `Q` se van generando números enteros desde 0 a `length (L) - 1`. Se extrae el elemento j-ésimo de `l` y se inserta en `Q`. Finalmente, se vuelven a pasar todos los elementos de la cola `Q` a la lista `L`. [Ejercicio tomado en el Exámen Final del 05/07/01]

Solución

```
1  void random_shuffle(list<int> &L) {
2    list<int>::iterator p,z;
3    list<int> q;
4    int k,n;
5    // Cuenta el numero de elementos en la lista L
6    n=0;
7    p=L.begin();
8    while (p++!=L.end()) n++;
9    // En cada iteracion del lazo se saca un elemento
10   // al azar de la lista L y se lo inserta en la cola Q
11   for (int h=n;h>0;h--) {
12     // A esta altura la lista L tiene m elementos
13     // asi que generamos un entero entre 0 y m-1
14     k=irand(h);
15     // nos posicionamos en el elemento k en la lista L
16     p=L.begin();
17     for (int j=0;j<h-1;j++) p++;
18     // inserta el elemento k de la lista L al final de la cola Q
19     // y lo elimina de la lista L
20     q.insert(q.end(),*p);
21     L.erase(p);
22   } // end h
23   // Vuelve a copiar todos los elementos de la cola a la lista
24   z=q.begin();
25   while (z!=q.end()) {
26     L.insert(L.end(),*z);
27     z=q.erase(z);
28   } // end while
29 }
```

Resuelto en el archivo `random_shuffle.cpp`

Ejercicio 14 Dada una lista de enteros `L` y dos listas `SEQ` y `REEMP` escribir una función `void reemplaza (list<int> &L, list<int> &SEQ, list<int> &REEMP)` que busca todas las secuencias de `SEQ` en `L` y las reemplaza por `REEMP`. Por ejemplo, si L=(1 2 3 4 5 1 2 3 4 5 1 2 3 4 5), SEQ=(4 5 1) y REEMP=(9 7 3), entonces despues de llamar a `reemplaza` debe quedar L=(1 2 3 9 7 3 2 3 9 7 3 2 3 4 5). Este procedimiento tiene un efecto equivalente a la función `reemplazar` de los editores de texto. [Tomado el 1er parcial, 16 abril 2002]

Solución

```
1  void reemplaza1 (list <int> & L,
2                   list <int> & SEQ,
3                   list <int> & REEMP)  {
4    queue<int> C ;
5    list<int> :: iterator p, q, r;
6    list<int> H ;
7    // copia REEMP en la lista auxiliar H
8    H.clear ();
9    r = REEMP.begin () ;
10   while (r != REEMP.end () ) { H.insert (H.begin (),*r) ; r++ ; }
11   // Inicializa posiciones
12   p =   L.begin ();
13   q = SEQ.begin ();
14   // recorre lista L
15   while ( p != L.end() ) {
16     if ( q == SEQ.end () ) {
17       // 1) llegamos al fin de SEQ pues todo SEQ esta en la lista L
18       //    vaciamos la cola C
19       while ( !C.empty () ) C.pop ();
20       //    insertamos REEMP en la lista L
21       r = H.begin () ;
22       while (r != H.end () ) { p = L.insert (p++,*r); r++ ; }
23       // reseteamos la posicion Q en la lista SEQ
24       q = SEQ.begin ();    }
25     else if (*p == *q) {
26       // 2) Hay otro elemento en la lista L igual al de SEQ
27       //    por lo que lo ponemos en la cola C
28       C.push (*p) ;
29       //    lo suprimimos en L y avanzamos en SEQ
30       p = L.erase (p);
31       q++ ;            }
32     else {
33       // 3) los elementos no son iguales
34       while ( !C.empty () ) {
35         p = L.insert ( p, C.front () );
36         C.pop ();
37         p++;
38       } // end while
39       p++;
40     } // end if
41   } // while
42   while ( !C.empty() ) {
43     p = L.insert ( p, C.front () );
44     C.pop ();
45     p++;
46   } // end while
47 }
48
49 // ------------------------------------------------------------------
50 // Variante sin el uso de una cola
51 // Pero, como siempre, sin usar el operador "--"
52 //
53 // Se recorre mientras no sea fin de la lista L con el iterador P y,
54 // cada vez que aparezca el primer elemento de SEQ, se la registra
55 // con la posicion auxiliar Q. Luego copiamos P en T, y vamos
56 // avanzando T y Q simultaneamente mientras, o bien no lleguemos
57 // al final de L o de SEQ, o bien cuando no esta todo SEQ en el
58 // tramo analizado. Si toda la secuencia SEQ se encontro en ese
59 // tramo, entonces inicializamos T en Q (en donde empezaba SEQ)
60 // y R en comienzo de REEMP. Vamos suprimiendo el elemento de L
61 // apuntado por T, insertamos el apuntado por R y avanzamos T y R.
62 // Finalmente actualizamos P (el cual recorre L actual) y seguimos
63 // explorando.
```

```
64 void reemplaza2 ( list <int> & L,
65                    list <int> & SEQ,
66                    list <int> & REEMP){
67   list <int> :: iterator p, q, r, t;
68   bool esta ;
69   p = L.begin ();
70   while ( p != L.end () ) {
71     r = SEQ.begin () ;
72     q = p ;          // recordamos donde empezaria (quizas) SEQ en L
73     t = p ;          // inicializamos T en P
74     esta = true ; // somos optimistas
75     while ( r != SEQ.end () && t != L.end () ) {
76       if  ( *r == *t ) { // seguimos encontrando SEQ en L
77         r++ ;
78         t++ ; }
79       else {
80         esta = false ;   // la secuencia SEQ es incompleta
81         break ;
82       } // end if
83     } // end while
84     if (esta == true) {
85       r = REEMP.begin ();
86       t = q ;               // inicia en el lugar donde empezo SEQ
87       while ( r != REEMP.end () ) { // borra e inserta
88         t = L.erase (t) ;  // suprimimos y actualizamos
89         L.insert (t, *r) ; // inserta un elemento de SEQ en L
90         r++ ;              // avanzo en SEQ
91       } // end  while
92       p = t ;              // actualizo iterador del lazo principal
93     } // end if
94     p++ ;                  // avanzo
95   } // end while
96 } // end void
```

Resuelto en el archivo `reemplaza.cpp`

Ejercicio 15 (a) Escriba una función `void refina (list<double> &L, double delta)` tal que dada una lista inicial de reales clasificados de menor a mayor `L`, refina inserta elementos entre los de `L`, de tal modo que la diferencia máxima entre elementos de la lista final sea menor o igual que delta; (b) Escriba una función `void desrefina (list<double> &L, double delta)` tal que dada una lista inicial de reales clasificados de menor a mayor `L`, desrefina suprime elementos de `L`, de tal modo que la diferencia minima entre elementos de la lista final sea mayor o igual que delta.

Solución

```
1  void refina (list<double> &L, double delta) {
2    list<double>::iterator p;
3    double x, y ;
4    p = L.begin ();
5    while (p != L.end () ) {
6      x = (*p) + delta; p++ ;
7      if (p == L.end () ) break;
8      y = (*p) ;
9      while (x < y) {
10       p = L.insert (p,x); p++;
11       x = x + delta ;
12     } // end while
13   } // end while
14 }
15
16 // ---------------------------------------------------------------------
17 void desrefina (list<double> &L, double delta) {
18   list <double> :: iterator p, q ;
19   p = L.begin () ;
20   while (p != L.end ()) {
21     q = p ; q++ ;
22     while (q != L.end () && (*q) < (*p) + delta ) {
23       q = L.erase (q) ;
24     } // end while
25     p++;
26   } // end while
27 }
```

Resuelto en el archivo `refina.cpp`

Ejercicio 16 Usando las operaciones del TAD lista, escribir una función `void rejunta (list<int> &L, int A)` que, dada una lista de enteros `L`, agrupe elementos de tal manera que en la lista queden solo elementos mayores o iguales que `A`. El algoritmo recorre la lista y, cuando encuentra un elemento menor, empieza a agrupar el elemento con los siguientes hasta llegar a `A` o hasta que se acabe la lista. Por ejemplo, si L=[3,4,2,4,1,4,4,3,2,2,4,1,4,1,4,4,1,4,4,2], entonces rejunta (L,10) da L=[13,12,13,10,10]. En la lista final NO deben quedar elementos menores que `A` salvo, eventualmente, el último. [Ejercicio tomado en el Exámen Final del 05/07/01]

Solución

```
1  void rejunta (list<int> &L, int a) {
2    list<int>::iterator p ;
3    int suma ;
4    p = L.begin ();
5    while ( p != L.end () ) {
6      if (*p > a)
7        p++ ;
8      else {
9        suma = 0 ;
10       while ( p != L.end () && suma < a) {
11         suma = suma + (*p) ;
12         p = L.erase (p);
13       } // end while
14       p = L.insert (p, suma);
15       p++;
16     } // end if
17   } // end while
18 }
```

Resuelto en el archivo `rejunta.cpp`

Ejercicio 17 Escriba procedimientos para insertar, suprimir y buscar un elemento en una lista ordenada `L`. Versión **sin** `funciones genéricas` (comparar con `sorted_list2.cpp` y `sorted_list3.cpp`).

Solución

```
1  // inserta un item "x" en la lista ordenada "L"
2  void inserta_ord (list<int> & L, const int & x) {
3    list<int>::iterator p ;
4    p = L.begin ();
5    while (p != L.end () && *p < x) p++;      // avanza si *p < x
6    L.insert (p, x);
7  } // end void
8
9  // ------------------------------------------------------------------
10 // suprime un item "x" en la lista ordenada "L"
11 void suprime_ord (list<int> & L, const int & x) {
12   list<int>::iterator p ;
13   p = L.begin ();
14   while (p != L.end () && *p < x) p++;         // avanza si *p < x
15   if (p != L.end () && *p == x) L.erase (p);  // suprime "p"
16 } // end void
17
18 // ------------------------------------------------------------------
19 // busca posicion "p" de un item "x" en la lista ordenada "L"
20 list<int>::iterator busca_ord (list <int> & L, const int & x) {
21   list<int>::iterator p ;
22   p = L.begin ();
23   while (p != L.end() && *p < x) p++;  // avanza si *p < x
24   if ( p != L.end () && *p == x )
25     return p ;                         // la posicion de "x" en "L"
26   else
27     return L.end ();                   // retorna fin de lista
28 } // end iterator
```

Resuelto en el archivo `sorted_list1.cpp`

Ejercicio 18 Escriba procedimientos para insertar, suprimir y buscar un elemento en una lista ordenada L. Versión **únicamente** con `funciones genéricas` (comparar con `sorted_list1.cpp` y `sorted_list3.cpp`).

Solución

```
1  // inserta un item "x" en la lista ordenada "L"
2  template <typename T>
3  void inserta_ord (list <T> & L, const T & x) {
4    typename list<T>::iterator p ;
5    p = L.begin ();
6    while (p != L.end () && *p < x) p++;     // avanza si *p < x
7    L.insert (p, x);
8  } // end void
9
10 // ------------------------------------------------------------------
11 // suprime un item "x" en la lista ordenada "L"
12 template <typename T>
13 void suprime_ord (list <T> & L, const T & x) {
14   typename list<T>::iterator p ;
15   p = L.begin ();
16   while (p != L.end () && *p < x) p++;        // avanza si *p < x
17   if (p != L.end () && *p == x) L.erase (p);  // suprime "p"
18 } // end void
19
20 // ------------------------------------------------------------------
21 // busca posicion "p" de un item "x" en la lista ordenada "L"
22 template <typename T>
23 typename list<T>::iterator busca_ord (list <T> & L, const T & x) {
24   typename list<T>::iterator p ;
25   p = L.begin ();
26   while (p != L.end() && *p < x) p++;  // avanza si *p < x
27   if ( p != L.end () && *p == x )
28     return p ;                         // la posicion de "x" en "L"
29   else
30     return L.end ();                   // retorna fin de lista
31 } // end iterator
```

Resuelto en el archivo `sorted_list2.cpp`

Ejercicio 19 Escriba procedimientos para insertar, suprimir y buscar un elemento en una lista ordenada L. Versión mediante una `clase genérica` (comparar con `sorted_list1.cpp` y `sorted_list2.cpp`).

Solución

```
template<typename T>  // sintaxis para una clase generica:
class sorted_list {   // template <class Tipo> class Nombre { }
private:
  list<T> data;
public:
  typedef typename list<T>::iterator posicion;
  sorted_list () {}
  sorted_list (list<T> & L) : data(L) {this->data.sort();}
  ~sorted_list () { }
  void clear() { this->data.clear(); }
  posicion begin () {return this->data.begin();};
  posicion end () {return this->data.end();}
  posicion insert (const T &); // no hay posicion pues esta ordenada
  posicion erase (const T &);
  posicion find (const T &);
};

// ------------------------------------------------------------------
// inserta un item "x" en la lista ordenada "L"
template<typename T>
typename sorted_list<T>::posicion sorted_list<T>::insert(const T& x){
  posicion p = begin();
  while ( p != end() && *p < x ) p++; // avanzar
  p = this->data.insert (p,x); // inserta item 'x' en 'p'
  return p;
}

// ------------------------------------------------------------------
// suprime un item "x" en la lista ordenada "L"
template<typename T>
typename sorted_list<T>::posicion sorted_list<T>::erase(const T& x) {
  posicion p = begin();
  while ( p != end() && *p < x ) p++; // avanzar mientras pueda
  if ( p != end() ) {           // si "p" no es final de "L"
    if ( *p == x )              // y como "*p" es igual a "x", borrar
      p = this->data.erase (p); // y refrescar posicion
    else                        // como "x" no esta en posicion "p"
      p++;                      // avanza
  }
  return p;
}

// ------------------------------------------------------------------
// busca posicion "p" de un item "x" en la lista ordenada "L"
template<typename T>
typename sorted_list<T>::posicion sorted_list<T>::find(const T& t) {
  posicion p = begin();
  while ( p!= end() && *p < t ) p++; // avanzar
  if( p != end() && *p == t ) // '*p' esta en 'L', retornar 'p'.
    return p;
  else                        // '*p' no esta en 'L', retornar 'end()'.
    return end();
}
```

Resuelto en el archivo `sorted_list3.cpp`